In [1]:
```
import pandas as pd

train = pd.read_csv("train.csv")

train.head()
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [2]:
```
train.shape
```

Out[2]:
```
(614, 13)
```

In [5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [7]:
```
# checking for missing values

train.isna().sum()
```

Out[7]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [9]:
```
train.describe()
```

Out[9]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

In [11]:
```
train.describe(include=[object])
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area
count	614	601	611	599	614	582	614
unique	614	2	2	4	2	2	3
top	LP001002	Male	Yes	0	Graduate	No	Semiurban
freq	1	489	398	345	480	500	233

In [13]:
```
# dropping PassengerId, Name and Ticket

train = train.drop(['Loan_ID'], axis=1)
```

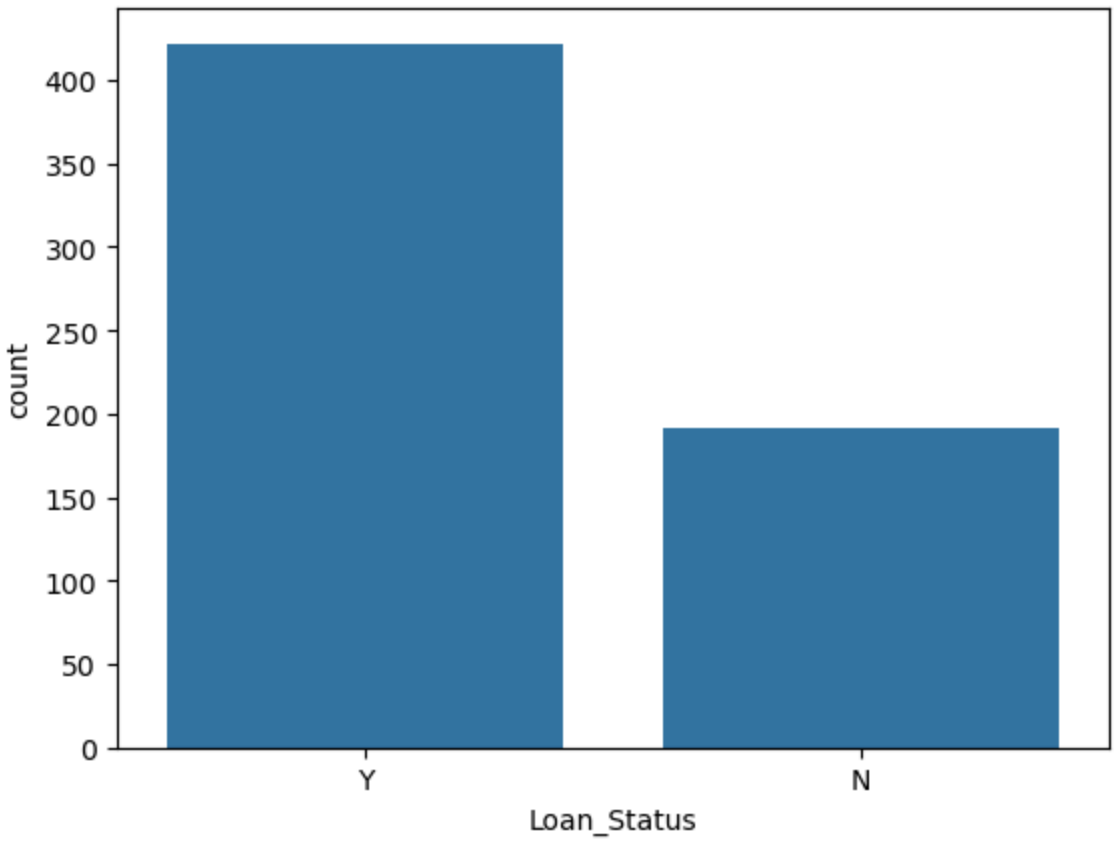
In [15]:
```
train['Loan_Status'].value_counts()
```

Out[15]:
```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

In [17]:
```
import seaborn as sns

sns.countplot(x=train['Loan_Status'])
```

Out[17]: <Axes: xlabel='Loan_Status', ylabel='count'>

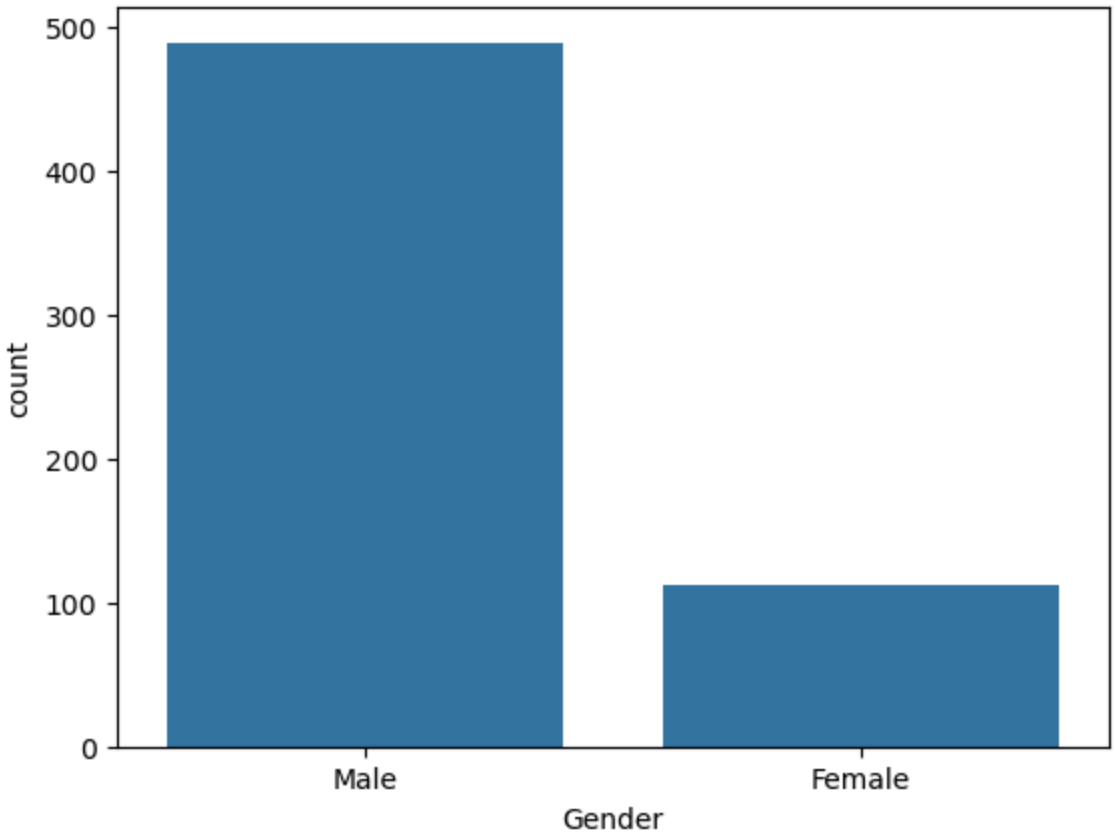


In [19]:
```
train['Gender'].value_counts()
```

Out[19]:
```
Gender
Male      489
Female    112
Name: count, dtype: int64
```

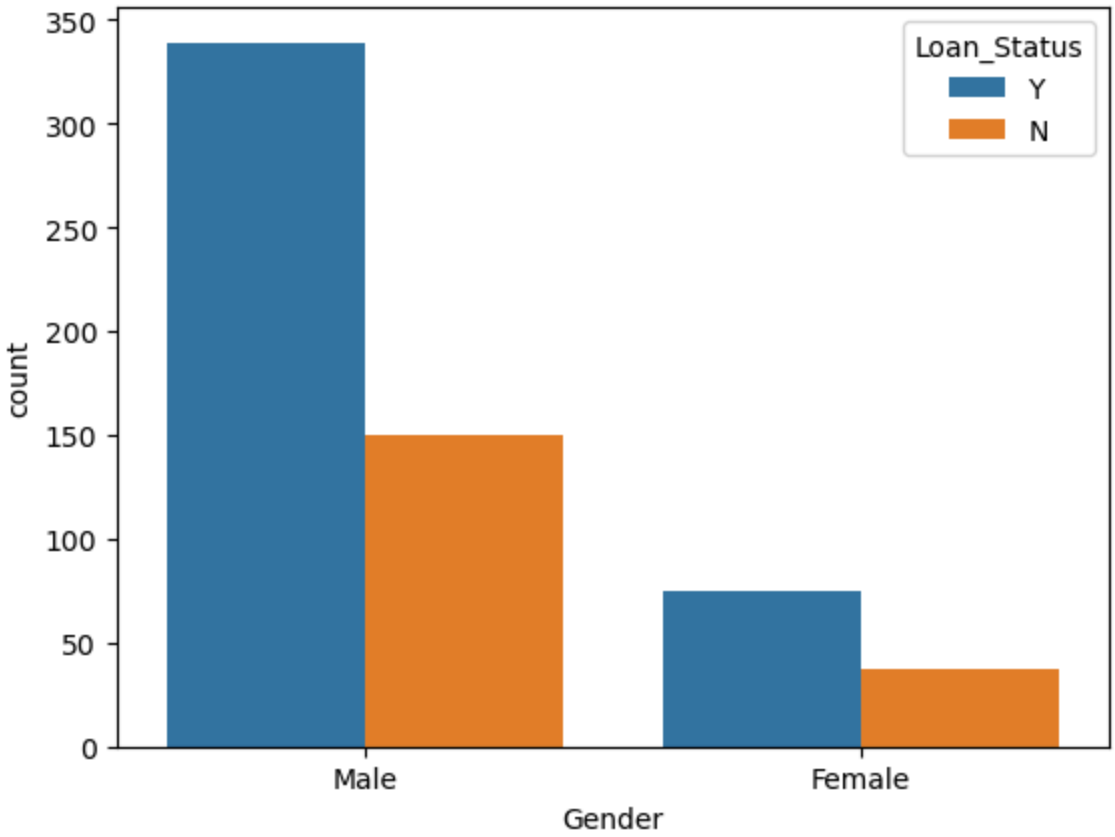
In [21]:
```
sns.countplot(x=train['Gender'])
```

Out[21]: <Axes: xlabel='Gender', ylabel='count'>



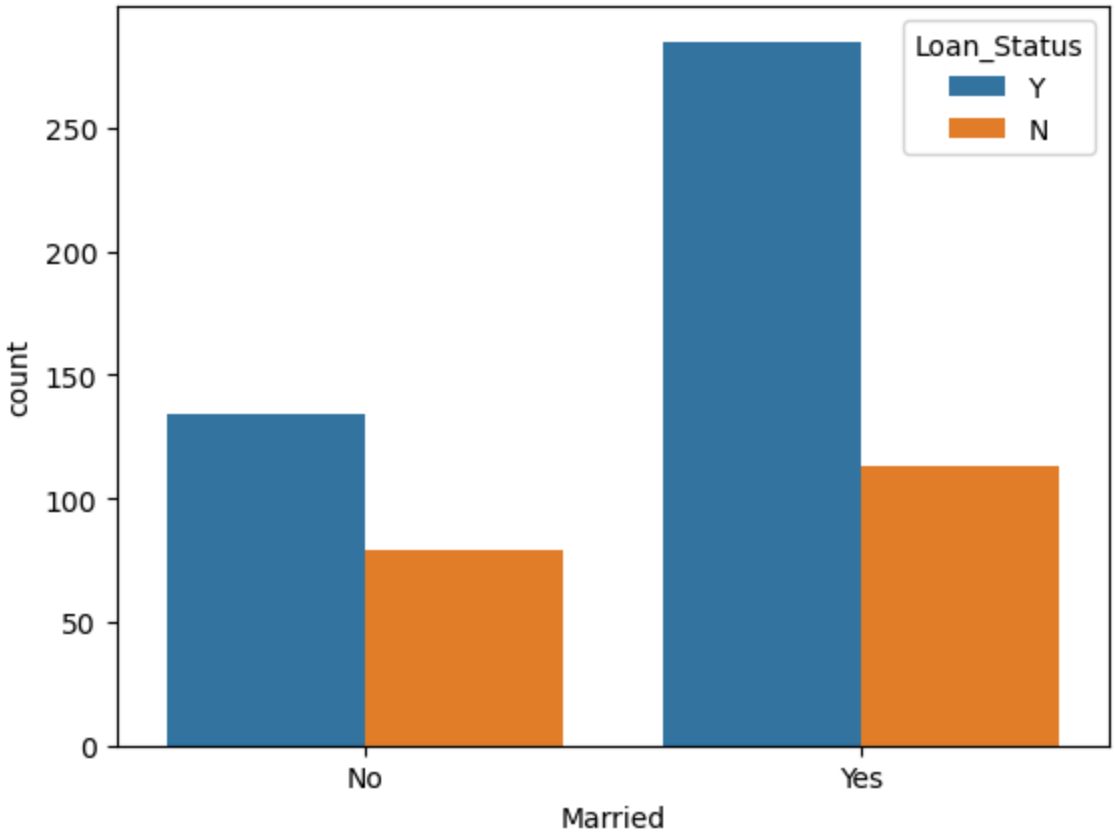
In [23]:
```
sns.countplot(x=train['Gender'], hue=train['Loan_Status'])
```

Out[23]: <Axes: xlabel='Gender', ylabel='count'>



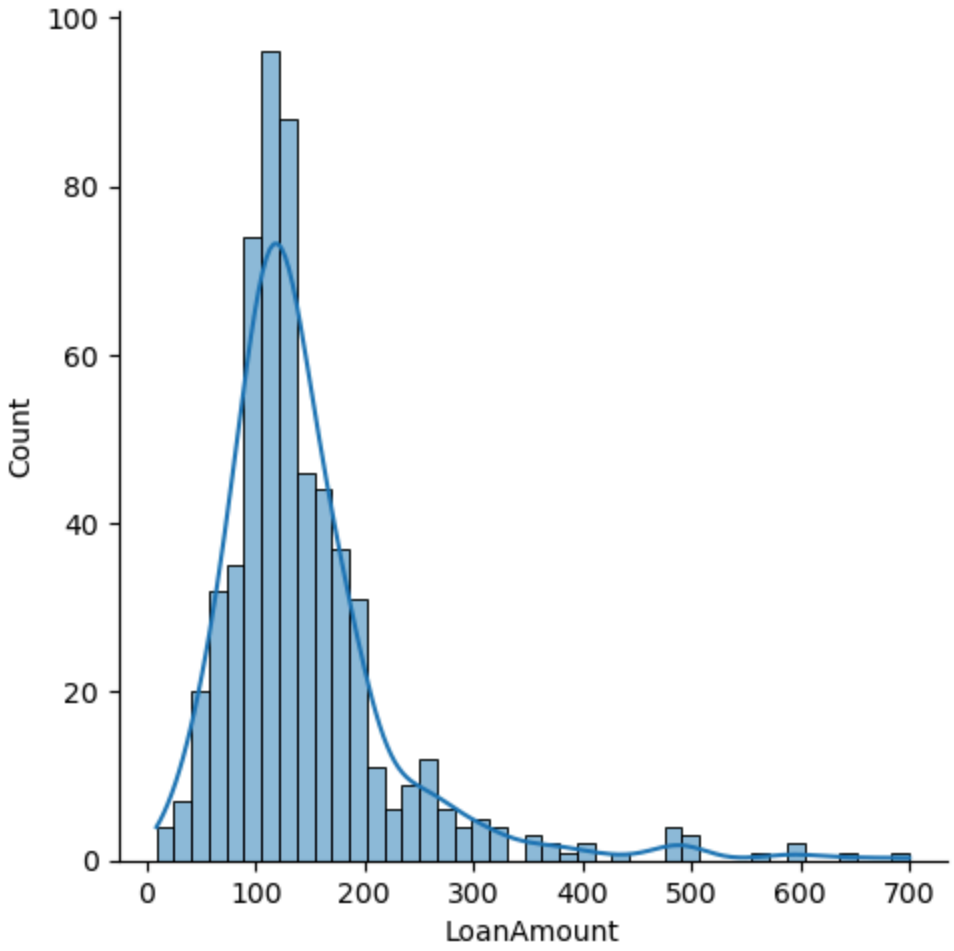
In [25]:
```
sns.countplot(x='Married', data=train, hue='Loan_Status')
```

Out[25]:
```
<Axes: xlabel='Married', ylabel='count'>
```



In [27]:
```
sns.displot(train['LoanAmount'], kde=True)
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x2477fad2f00>
```



In [29]:
```
train['LoanAmount'].skew()
```

Out[29]:
```
2.677551679256059
```

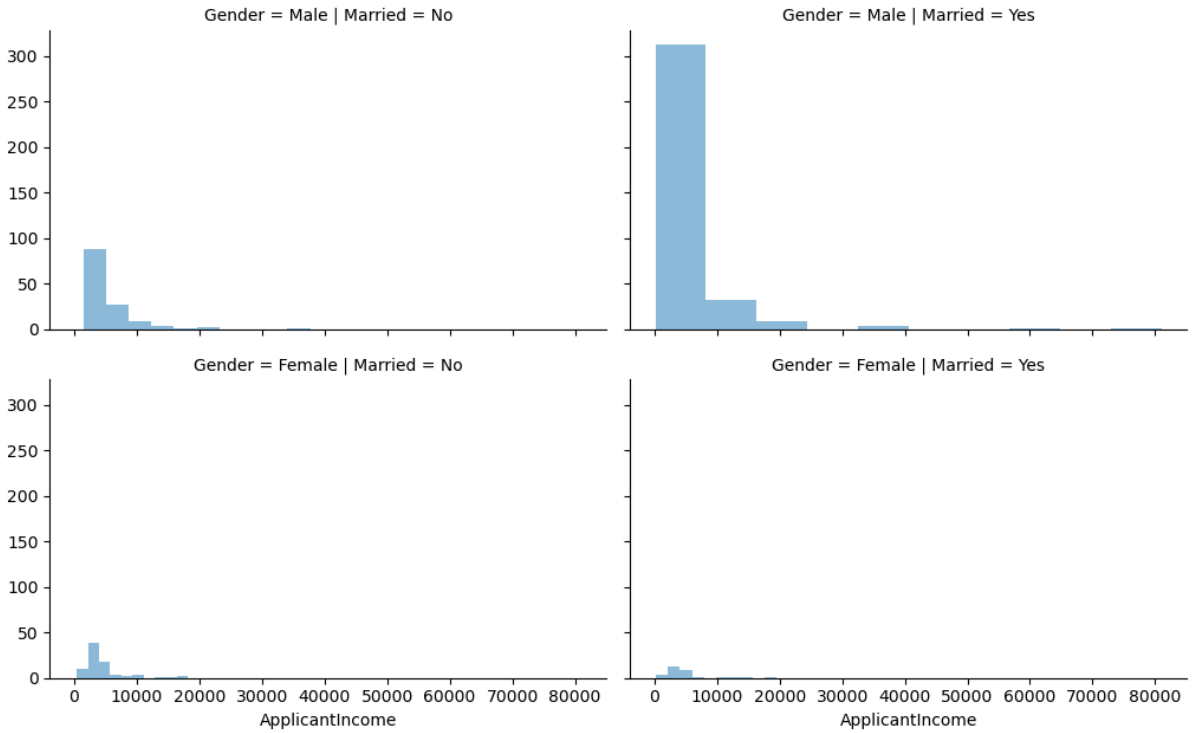
In [31]:
```
import matplotlib.pyplot as plt

grid = sns.FacetGrid(train, row='Gender', col='Married', height=3.2, aspect=1.6)

grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)

grid.add_legend()
```

Out[31]:
```
<seaborn.axisgrid.FacetGrid at 0x2477faa8b60>
```



In [33]:
```
rplot(train, hue='Loan_Status', height=2.5)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 rplot(train, hue='Loan_Status', height=2.5)

NameError: name 'rplot' is not defined
```

In [35]:
```
train.isna().sum()
```

Out[35]:
```
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

In [43]:
```
train['Gender'] = train['Gender'].fillna(train['Gender'].mode()[0])
```

In [45]:
```
train['Married'] = train['Married'].fillna(train['Married'].mode()[0])
```

In [41]:
```
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
```

```
C:\Users\Vaish\AppData\Local\Temp\ipykernel_24124\869824710.py:1: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
```

In [47]:
```
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
```

```
C:\Users\Vaish\AppData\Local\Temp\ipykernel_24124\1340705068.py:1: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
```

In [49]:
```
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

```
C:\Users\Vaish\AppData\Local\Temp\ipykernel_24124\3228316714.py:1: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

In [51]:
```
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
```

```
C:\Users\Vaish\AppData\Local\Temp\ipykernel_24124\2271678739.py:1: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=Tru
e)
```

In [53]:
```
train['Credit_History'].skew()
```

Out[53]:
```
-1.8823610612186696
```

In [55]:
```
train.isna().sum()
```

Out[55]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

In [57]:
```
from sklearn.preprocessing import LabelEncoder

feature_col = ['Gender','Married','Dependents','Education','Self_Employed','Propert

le = LabelEncoder()

for col in feature_col:

    train[col] = le.fit_transform(train[col])
```

In [59]:
```
train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N" : 0})
```

```
C:\Users\Vaish\AppData\Local\Temp\ipykernel_24124\1785350424.py:1: FutureWarning: Do
wncasting behavior in `replace` is deprecated and will be removed in a future versio
n. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. T
o opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting',
True)`
  train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N" : 0})
```

In [61]:
```
train.head(3)
```

Out[61]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	1	0	0	0	0	5849	
1	1	1	1	0	0	4583	
2	1	1	0	0	1	3000	

In [64]:
```
train['total_income'] = train['ApplicantIncome'] + train['CoapplicantIncome']
```

In [66]:
```
train.drop(columns=['ApplicantIncome', 'CoapplicantIncome'], inplace=True)
```

In [68]:
```
train.head(3)
```

Out[68]:

	Gender	Married	Dependents	Education	Self_Employed	LoanAmount	Loan_Amount_T
0	1	0	0	0	0	128.0	3
1	1	1	1	0	0	128.0	3
2	1	1	0	0	1	66.0	3

In [70]:
```
train.columns
```

Out[70]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status', 'total_income'],
      dtype='object')
```

In [72]:
```
rel_feat =['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',

           'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'to
```

In [74]:
```
rel_feat_corr = train.corr()['Loan_Status'][['Gender', 'Married', 'Dependents', 'Ed

                                             'Self_Employed', 'LoanAmount', 'Loan_A

                                             'Credit_History', 'Property_Area', 'to
```

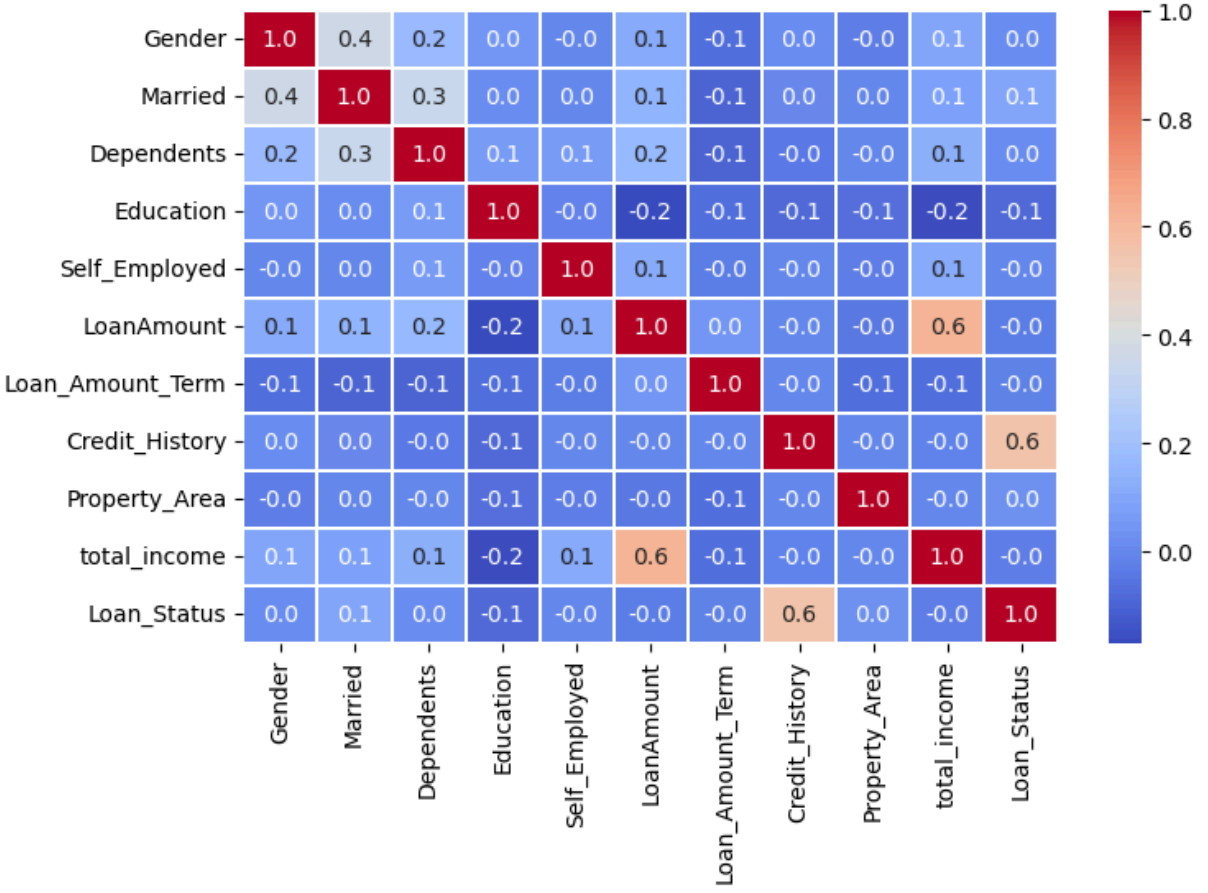
In [76]:
```
rel_feat_corr
```

Out[76]:
```
Gender              0.017987
Married             0.091478
Dependents          0.010118
Education          -0.085884
Self_Employed      -0.003700
LoanAmount         -0.033214
Loan_Amount_Term   -0.022549
Credit_History      0.561678
Property_Area       0.032112
total_income       -0.031271
Name: Loan_Status, dtype: float64
```

In [78]:
```
plt.figure(figsize=(8,5))

sns.heatmap(train[rel_feat].corr(), cmap='coolwarm', annot=True, fmt='.1f', linewid

plt.show()
```



In [80]:
```
#Separating target variable and other variables

X=train.drop(columns='Loan_Status')

y=train['Loan_Status']
```

In [82]:
```
#Splitting the data into train and test sets

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.30, random_state=7
```

In [84]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, max_features='sqrt')

rf = rf.fit(X_train, y_train)

rf_pred=rf.predict(X_test).astype(int)
```

In [86]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

print(confusion_matrix(y_test,rf_pred))

print(classification_report(y_test,rf_pred))

print("Accuracy:",accuracy_score(y_test, rf_pred))
```

```
[[ 31  30]
 [  9 115]]
              precision    recall  f1-score   support

           0       0.78      0.51      0.61        61
           1       0.79      0.93      0.86       124

    accuracy                           0.79       185
   macro avg       0.78      0.72      0.73       185
weighted avg       0.79      0.79      0.78       185

Accuracy: 0.7891891891891892
```

In [ ]: